PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-163625
(43)Date of publication of application : 09.06.1992

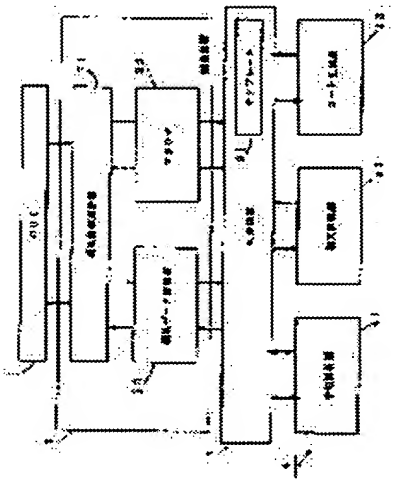
(51)Int.Cl. G06F 9/06

(21)Application number : 02-291013
(22)Date of filing : 29.10.1990

(71)Applicant : FUJI XEROX CO LTD
(72)Inventor : IWATA MASATAKE
KURABE ATSUSHI
ICHIJI HIROSHI

(54) SOURCE CODE GENERATION DEVICE

(57)Abstract:
PURPOSE: To control information with much versality by absorbing the difference of the output format of a source code based on the difference of GUI construction surroundings by means of a template.
CONSTITUTION: A system consists of GUI (graphic user interface) 1, an editing device 2, a storage device 3 and a source code generation device 4. The template 31 where a character string which is directly added as the source code being the object of generation and information which requires correction to a content whose construction is analyzed for making the source code being the object of generation are described is provided. Different code generation elements between different GUI construction surroundings are absorbed through the template 31. Thus, the source code of the target program can conventionally be generated for various GUI construction surroundings.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

⑲日本国特許庁（JP）　⑪特許出願公開

⑫公開特許公報（A）　平4－163625

⑤Int. Cl.[5]	識別記号	庁内整理番号	㊸公開　平成4年(1992)6月9日
G 06 F 9/06	430 E	7927－5B	
	430 J	7927－5B	
	430 M	7927－5B	

審査請求　未請求　請求項の数 3　（全18頁）

㊹発明の名称　ソースコード生成装置

㉑特　願　平2－291013
㉒出　願　平2(1990)10月29日

㉗発明者　岩田　正武　神奈川県川崎市高津区坂戸100番1号　KSPR&Dビジネスパークビル　富士ゼロックス株式会社内
㉗発明者　倉部　淳　神奈川県川崎市高津区坂戸100番1号　KSPR&Dビジネスパークビル　富士ゼロックス株式会社内
㉗発明者　伊知地　宏　神奈川県川崎市高津区坂戸100番1号　KSPR&Dビジネスパークビル　富士ゼロックス株式会社内
㉗出願人　富士ゼロックス株式会社　東京都港区赤坂3丁目3番5号
㉔代理人　弁理士　木村　高久

明　細　書

1．発明の名称

ソースコード生成装置

2．特許請求の範囲

（1）複数のグラフィカル・ユーザインターフェース部品がその属性に従って分類された木構造のデータとして表現されるグラフィカル・ユーザインターフェースデータから、特定プログラミング言語のソースコードを生成するソースコード生成装置であって、

前記グラフィカル・ユーザインターフェースデータを、その構成部品の種類、構成部品の属性名、及び構成部品の属性値からなるトークンの列に分解する字句解析手段と、

この分解されたトークンの列に基づき、部品の識別子、部品の名前及び種類、リソースリスト、及びオブジェクトプログラムで使用する変数名によって構成されるデータテーブルを生成する構文解析手段と、

前記生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために前記生成されるデータテーブルに対して施う必要のある情報が記述されたテンプレートと、

前記生成されるデータテーブルを、前記テンプレートに記述されている情報に基づき変換しかつ、同テンプレートにより指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段と、

を具えるソースコード生成装置。

（2）前記構文解析手段は、前記部品の名前と前記部品の識別子とをセパレータにて結合して前記オブジェクトプログラムで使用する変数名を生成する

請求項（1）記載のソースコード生成装置。

（3）前記コード生成手段は、

前記生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数の

サブコード生成手段と、

これら各サブコード生成手段にて生成される各別のコードを、対象とするプログラミング言語の仕様に応じて結合するコードマージ手段と、

を具えて構成され、

前記テンプレートは、前記サブコード生成手段の別に複数用意される

請求項（１）記載のソースコード生成装置。

３．発明の詳細な説明

［産業上の利用分野］

この発明は、ワークステーションやパーソナルコンピュータ等を通じて設計、編集されたグラフィカル・ユーザインターフェースデータから特定プログラミング言語のソースコードを生成するに好適なソースコード生成装置に関する。

［従来の技術］

ワークステーションやパーソナルコンピュータ等に視覚的な操作環境を提供するグラフィカル・ユーザインターフェース（以下、ＧＵＩと略称する）は、これらワークステーションやパーソナルコンピュータ等を通じての知的生産を著しく高め得る手段或いは技術として、近年、大いに注目を集めている。

例えば、

◆「ＧＭＷ（ギブ・ミー・モア・ウィンドウズ）ウィンドウ・システム上のアプリケーション構築について」大谷浩司　他著　コンピュータソフトウェア

Vol.17　No.1　(1990.1.16) P.45-60

◆「ヒューマンインタフェースの最先端」日本ソフトウェア科学会　平成２年１月１８日発行

P.29-48「グラフィカルなユーザ・インターフェースとその開発環境について」萩谷昌己著

◆「User Interface Management Systems（ユーザ・インターフェース・マネージメント・システムズ）」G.E.Pfaff（ジー・イー・ファフ）編

Springer-Verlag（スプリンガー出版）

1983.11.1-3

P.9-20「Report on Dialogue Specification Tools（リポート・オン・ダイアログ・スペシフィケーション・ツールズ）」M.Green（エム・グリーン）著

◆「Human-Computer Interface Development: Concepts and Systems for Its Management（フューマン－コンピュータ・インターフェース・デベロップメント：コンセプト・アンド・システムズ・フォー・イッツ・マネージメント）」

H.REX HARTSON（エイチ・レックス・ハートソン）

DEBORAH HIX（デボアー・ヒックス）共著

ACM Computing Surveys　Vol.21　No.1　1989.3

P.5-92

等々は何れも、ＧＵＩについて考察された文献であり、この他にも、様々な形で、ＧＵＩに関する研究や開発が進められている。

ところで、このようなＧＵＩを実現するためのオブジェクトプログラムは通常、

(A) ＧＵＩ部品が所定に組み合わされたデータとして構成されるＧＵＩデータをＧＵＩ編集装置を通じて設計、編集する。

(B) この設計、編集されたＧＵＩデータをソースコード生成装置に読み込んで、特定のプログラミング言語のソースコードを生成する。

(C) この生成されたソースコードを、上記特定のプログラミング言語用のコンパイラによってコンパイルする。

といった手順を経て生成されるが、上記のＧＵＩ編集装置やソースコード生成装置は元来、ＯＳ（オペレーティングシステム）やプログラミング言語、ライブラリ等々のＧＵＩを構築する環境に大きく依存しているのが普通であることから、特にソースコード生成装置に限っていっても、

(1) 上記ＧＵＩを構築する環境が異なれば、ソースコードに埋め込まれるテキストやその形式、順序等も異なってくることから、これらＧＵＩの構築環境毎にソースコード生成装置を用意する必要がある。因みに従来は、ソースコードの出力フォーマットが同装置のソースコード生成プログラムに内蔵されているのが普通である。

(2) 上記オブジェクトプログラム内で使用する変

数名もこのソースコード生成装置を通じて自動的に生成されるが、生成されたソースコードを後に人為的に編集しようとした場合、この変数名がどのような目的で使用されているか理解しにくい。すなわち、こうしたソースコード生成装置では、人間が使用しないような文字列の組合せを選んで上記の変数名を自動生成することから、変数名の重複はないものの、その可読性は著しく低いものとなっている。

等々、実用上は、また汎用性を考える場合には、尚多くの問題を抱えている。

なお、上記ソースコード生成装置やコンパイラの概念については、例えば

◆「コンパイラの理論と実現」疋田、石畑　共著　共立出版　1989年 6月

に詳しい。

［発明が解決しようとする課題］

上述のように、従来のＧＵＩ開発環境におけるソースコード生成装置あっては、上記(1)～(2)として挙げたような課題を残すものであり、特に現在開発中のものも含む多くのＧＵＩ開発環境に対応すべく、その汎用性、並びに実用性を考える上では、これら課題の早急な解決が望まれる。

この発明は、こうした実情に鑑みてなされたものであり、いかなるＧＵＩ開発環境にあっても、上記の課題を解決して、汎用性に富み、可読性の高いソースコードを生成することのできるソースコード生成装置を提供することを目的とする。

［課題を解決するための手段］

こうした目的を達成するために、この発明では、複数のＧＵＩ部品がその属性に従って分類された木構造のデータとして表現されるＧＵＩデータから、特定プログラミング言語のソースコードを生成することを前提に、

(a) 前記ＧＵＩデータを、その構成部品の種類、構成部品の属性名、及び構成部品の属性値からなるトークンの列に分解する字句解析手段。

(b) この分解されたトークンの列に基づき、部品の識別子、部品の名前及び種類、リソースリスト、及びオブジェクトプログラムで使用する変数名によって構成されるデータテーブルを生成する構文解析手段。

(c) 前記生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために前記生成されるデータテーブルに対して補う必要のある情報が記述されたテンプレート。

(d) 前記生成されるデータテーブルを、前記テンプレートに記述されている情報に基づき変換しかつ、同テンプレートにより指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段。

を少なくとも具えてソースコード生成装置を構成するようにする。

［作用］

上記(c)の態様で必要文字列（テキスト）及び必要情報が記述されたテンプレートを用意することで、更にはまた、上記(d)の態様でこのテンプレートを利用するコード生成手段を具えることで、上記字句解析手段及び構文解析手段を通して生成されるデータテーブルであれば、ＧＵＩ構築環境がどのようなものであれ、その生成されるソースコードは、対象となるプログラミング言語の仕様に応じた所定の規則に従ったものとなる。すなわち、ＧＵＩ構築環境の違いに基づくソースコードの出力フォーマットの違いといったようなものは上記テンプレートによって吸収されるようになる。したがって、各ＧＵＩの構築環境毎に、それぞれその仕様に応じた各別のテンプレートを用意しておき、利用するＧＵＩ構築環境に応じて、それに見合ったテンプレートをその都度使い分けるようにすれば、ソースコード生成装置自体は只１つのもので済むようになる。

なおこれに併せ、上記構文解析手段において、部品の名前と部品の識別子とをセパレータ（例えば"＿"）にて結合して前記の変数名を生成するようにすれば、この生成される変数名を、非常に可読性の高いものとすることができるようにもなる。これによって、上記自動生成されるソースコードの後の編集やメンテナンスも容易となる。

また更に、上記コード生成手段を、
(d-1) 生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数のサブコード生成手段。
(d-2) これら各サブコード生成手段にて生成される各別のコードを、対象とするプログラミング言語の仕様に応じて結合するコードマージ手段。
を具えて構成し、上記テンプレートも、これらサブコード生成手段の別に複数用意するようにすれば、ソースコードの行の並びに対して独立にコードを生成することができるようになり、該ソースコード生成装置としての汎用性も更に高いものとなる。

［実施例］

第１図に、この発明にかかるソースコード生成装置の一実施例をＧＵＩ（グラフィカル・ユーザインターフェース）編集装置とともに示す。

この実施例では便宜上、ＧＵＩの部品として、汎用ＯＳであるＵＮＩＸ上でのＧＵＩ構築環境の１つとしてしられているＸウィンドウシステム・バージョン１１の、Ａｔｈｅｎａ　Ｗｉｄｇｅｔ（アテナ・ウィジェット）を使用し、また生成の対象とするソースコードとしては、Ｃ言語で記述されたソースプログラムとする場合について説明する。

はじめに、この実施例のソースコード生成装置に読み込まれるＧＵＩデータの形態について、その大本となる編集装置との兼ね合いのもとに説明する。

まず、第１図において、ＧＵＩ１は、表示装置（図示せず）の制御プログラムとして、この表示装置上にＡｔｈｅｎａ　Ｗｉｄｇｅｔで定義されたＧＵＩ編集用の画面を視覚的に表示制御するＸウィンドウシステムからなる。そしてこのＧＵＩ１では、これら表示情報に対するキーボードやマウスなどの入力装置（図示せず）を通じてユーザ入力があった場合、これをイベントとして受け取って、そのイベント内容を、編集装置２にメッセージとして通知する。編集装置２では、以下に説明する全ての処理が、このＧＵＩ１（Ｘウィンドウシステム）との間でのメッセージ通信を通じて実行されるようになる。第２図に、このＧＵＩ１による表示装置を通じての表示制御画面の一例を示す。

因みにこの第２図に示す画面において、１１は、ＧＵＩ部品（ｗｉｄｇｅｔ）表示用のウィンドウであるブラウザエリア、１２は、ＧＵＩデータの編集（設計）用のウィンドウである編集用エリア、１３ａ、１３ｂ及び１３ｃは、各々「ｅｄｉｔ（エディット）」メニュー、「ｆｉｌｅ（ファイル）」メニュー、及び「ｍａｉｎ（メイン）」メニューをプルダウン表示させるためのメニューボタン、１４は、上記ブラウザエリア１１に所望のＧＵＩ部品を表示させるためのブラウジングボタン、をそれぞれ示している。後に述べるＧＵＩデータの編集（設計）操作等は、全てこのＧＵＩ画面を通じて実行されるようになる。

また、第１図において、上記ＧＵＩ１を通じてユーザ入力によるイベントが通知される編集装置２は、大きくは、該通知されるイベントを都度の編集状態に応じて解析する部分である編集動作解析部２１と、上記ブラウザエリア１１を通じて操作されるＧＵＩ部品の管理を主に行う部分であるブラウザ２２と、上記編集用エリア１２を通じて設計或いは編集されるＧＵＩデータを管理する部分である編集データ管理部２３とを具えて構成される。

すなわちこの編集装置２では、例えば
(1) ブラウザ２２によって管理されているＧＵＩ部品が上記ブラウザエリア１１を通じて指定される。
(2) この指定された部品が該ブラウザエリア１１から上記編集用エリア１２の所定位置にマウスによりドラッグされるなどして配置される。
といったようなユーザによる編集操作に対し、
(3) 編集動作解析部２１が、この間のイベントを解析してその編集内容が「部品の追加」である旨判断し、該編集動作解析部２１から上記編集データ管理部２３に対して当該ＧＵＩ部品の追加指令を発する。

(4) これを受けた編集データ管理部２３が、その時点で管理している当該編集ＧＵＩデータにこの部品の情報を加えて、これを新たなＧＵＩデータとして管理する。

といったかたちで、ユーザによるＧＵＩ編集作業を裏面から支えるよう動作する。

また第１図において、記憶装置３は、１つには、上記編集装置２のブラウザ２２によって管理されるＧＵＩ部品をはじめ、同編集装置２を通じてこうして編集（設計）、管理されるＧＵＩデータを格納、保持するための装置である。この記憶装置３はまた、この実施例のソースコード生成装置における各種作業データの保管装置としても併せ用いられるが、その詳細については、ソースコード生成装置とともに後に詳述する。

ところで、上記編集装置２では、ＧＵＩを構成する各部品をオブジェクトとして捉えている。このため、これら部品をクラスという概念で分類することができるようになる。ここでいうクラスとは、同じ特徴を持つオブジェクトの仕様を定義したものである。

またここでは、オブジェクト指向における継承（ｉｎｈｅｒｉｔａｎｃｅ）の考えを取り入れている。これにより、上記クラスの定義を行うにも、その上位クラスとの差分のみを記述することで足りるようになる。

特に、この編集装置２として、上記のように、編集用エリア１２（第２図）に呼び出された各ＧＵＩ部品及びその編集内容をＧＵＩデータとして管理する部分である編集データ管理部２３では、これら各ＧＵＩ部品をこうしたオブジェクトとして捉えることにより、同部品をそれら各々の持つ属性に従ってカテゴリに分類し、木構造を用いて管理している。ここでのカテゴリは、上記クラスに対応し、下位のカテゴリの定義には、その上位のカテゴリとの差分のみが用いられる。

また、同編集データ管理部２３は、上記編集用エリア１２に呼び出された部品に対して、その各々に個別の識別子を設定する機能も併せ持つ。これら設定された識別子の値は、その都度、編集動作解析部２１に返されるとともに、該編集データ管理部２３によって管理されるＧＵＩデータ自身にも、それを構成する部品の識別情報として付加されるようになる。

第３図に、こうした編集装置２を通じて編集（設計）されるＧＵＩデータの一例を構造図として示す。

この第３図に例示するデータは、先の第２図においてその編集用エリア１２内に参考までに付記した部品をイメージしたものであって、例えば、識別子「６５２２」が付された「Ｗｉｎｄｏｗ（ウィンドウ）」部品ＷＷの上に置かれた

◇ｘ座標＝１００

◇ｙ座標＝２００

◇ラベル表示＝"ｃｏｍｍａｎｄ"

◇ｃａｌｌｂａｃｋ関数名＝ｆｏｏ

◇識別子＝６５２４

なる「Ｃｏｍｍａｎｄ（コマンド）」部品ＣＷを想定している。

そしてこの部品ＣＷは、同第３図に示されるように、１つのオブジェクトとして、大きくは「Ｐｒｏｐｅｒｔｙ（資源）」、「Ｉｄｅｎｔｉｆｉｅｒ（識別）」、及び「Ｓｔｒｕｃｔ（構造）」の各要素からなり、更にこれら各要素のうちの、「Ｐｒｏｐｅｒｔｙ」要素が、メジャーなカテゴリ、すなわち「ＭａｊｏｒＰｒｏｐｓ（メジャー資源）」として、上記位置の属性「ｘ－１００」及び「ｙ－２００」を含む「Ｇｅｏｍｅｔｒｙ（幾何）」情報と、マイナーなカテゴリ、すなわち「ＭｉｎａｒＰｒｏｐｓ（マイナー資源）」として、上記文字列及びｃａｌｌｂａｃｋ関数名の属性「ｌａｂｅｌ（ラベル表示）－Ｃｏｍｍａｎｄ」及び「ｃａｌｌｂａｃｋ（ｃａｌｌｂａｃｋ関数名）－ｆｏｏ」を含む情報と、からなり、「Ｉｄｅｎｔｉｆｉｅｒ」要素が、当該部品ＣＷの識別情報である「ｉｄ（識別子）－６５２４」、「ｎａｍｅ（部品の名前）－ｃｏｍｍａｎｄ」及び「ｃｌａｓｓ（クラス名）－Ｃｏｍｍａｎｄ」を含む情報からなり、「Ｓｔｒｕｃｔ」要素が、同部品ＣＷの置かれるいわば親部品としての上記

「Ｗｉｎｄｏｗ」部品ＷＷの識別子「ｐａｒｅｎｔ（親部品）＝６５２２」を含む情報からなる階層構造によって表現されている。

因みに、「Ｐｒｏｐｅｒｔｙ」要素において、上記位置の属性は全ての部品が持つものの、上記部品上に表示される文字列の属性等は全ての部品が持つとは限らないことから、すなわち、上記の例のような「コマンドボタン」の場合には、こうした文字列の属性を持つが、「スクロールバー」のような部品の場合には、こうした文字列の属性は持たないことから、上記位置の属性は、全ての部品に適用するメジャーなカテゴリとしてカテゴリ「ｍａｊｏｒＰｒｏｐｓ」に、他方の文字列等の属性は、全ての部品には適用しないマイナーなカテゴリとしてカテゴリ「ｍｉｎａｒＰｒｏｐｓ」に、それぞれ分類されている。

このように、上記編集装置２を通じて編集され、管理されるＧＵＩデータの情報は、第３図に示されるように、アイテムとそのアイテムの組によって記述される。そして上述のように、各アイテムは、類似した性質のもの同士がカテゴリにまとめられ、更に各カテゴリも、類似した種類のもの同士が上位のカテゴリにまとめられる階層構造となっている。この実施例においては、これをＳ式を用いて表記する。

Ｓ式とは、人工知能用言語である「ＬＩＳＰ（リスプ）」において多く用いられる「Symbolic expression（シンボリック・エクスプレッション）」の略である。このＳ式では、アイテムの構造を「かっこ（）」の組み合せを用いて表記するものであり、同じ階層にあるアイテムについは、これを同じ組の「かっこ（）」の中に記述し、それより１つ下の階層にあるアイテムについては、これを上位の「かっこ（）」の中に包含される別の組の「かっこ（）」の中に記述することで、その階層構造を表現するようにしている。

例えば、第３図に例示される上記の部品の情報は、該Ｓ式によって、

```
($Object

)                                    ...(1)
```

のように表記されることとなる。

このように、編集（設計）されるＧＵＩデータを、Ｓ式によって汎用的に表記される階層構造にを用いて管理し、保存するようにすることで、ＧＵＩ環境によって限定されない、より汎用性に富んだ情報管理が実現されるようになる。

なお、上記部品ＣＷの置かれるいわば親部品としての上記「Ｗｉｎｄｏｗ」部品ＷＷは、同Ｓ式によって

```
($Property
  ($MajorProps
    ($Geometry
      (@x $100)
      (@y $200)
    )
    ($MinarProps
      (@label $Command)
      (@callback $foo)
    )
  )
)
($Identifier
  (@id $6524)
  (@name $command)
  (@class $Command)
)
($Struct
  (@parent $6522)
)
```

```
($Object
  ($Property
    ($MajorProps
      ($Geometry
        (@x $0)
        (@y $0)
      )
      ($MinarProps
```

```
        ($Geometry
          (@width $300)
          (@height $400)
        )
      )
    )
  )
  ($Identifier
    (@id $8522)
    (@name $shell)
    (@class $Shell)
  )
  ($Struct
    (@parent NULL)
  )
)                                    ...（２）
```

のように表記されているとする。

さて、この実施例によるソースコード生成装置は、このようにＳ式によって表記され、記憶装置３上に保存されるＧＵＩ編集データを読み込んで、これからＣ言語で記述されたソースプログラムを生成するものである。

第１図に併せ示すように、このソースコード生成装置は、上記読み込んだデータを、部品の属性名、部品の属性値、及び部品のクラスからなるトークンの列に分解する字句解析部４１と、この分解されたトークンの列から、部品の識別子、部品のクラス名、部品の名前、部品の属性（リソースリスト）、及び生成目的とするプログラムで使用する変数名により構成されるデータテーブルを生成する構文解析部４２と、この生成されたデータテーブルを、上記記憶装置３上に予め用意されたテンプレート３１に記述されている情報に基づき変換しかつ、同テンプレート３１により指示される部位に埋め込みつつ、対象となるソースコードを生成するコード生成手段と、を具えて構成される。テンプレート３１には、生成対象とするソースコードとして直接に付加される文字列（テキスト）、及び同生成対象とするソースコードとするために上記生成されたデータテーブルに対して補う必要のある情報がそれぞれ記述されているものとする。

以下、これら各部の具体的な機能、並びに動作について、第４図～第１０図を併せ参照して順次詳述する。

まず、字句解析部４１では上記のように、（１）式、或いは（２）式として表記されるような編集データを記憶装置３から読み込んで、これをトークンの列に分解する。ここで分解されるトークンは、「ＬＢＲＡＮＣＨ（左かっこ）」、「ＲＢＲＡＮＣＨ（右かっこ）」、「ＫＥＹ（構造を示す名前）」、「ＬＩＴＥＲＡＬ（オブジェクトの属性値）」、及び「ＩＤＥＮＴ（オブジェクトの属性名）」を単位としたものである。

因みに上記（１）式或いは（２）式において、「＃」で始まる文字列は「ＫＥＹ（構造を示す名前）」であり、「@」で始まる文字列は「ＩＤＥＮＴ（オブジェクトの属性名）」であり、「＄」で始まる文字列は「ＬＩＴＥＲＡＬ（オブジェクトの属性値）」である。したがって、第４図に（Ａ）として示す上記（１）式にて表記される部品の場合には、この字句解析部４１による上記処理によって、同第４図（Ｂ）に示されるように、「（」、「＃Ｏｂｊｅｃｔ」、「（」、「＃Ｐｒｏｐｅｒｔｙ」、「（」、「＃ＭａｊｏｒＰｒｏｐｓ」、「（」、「＃Ｇｅｏｍｅｔｒｙ」、「（」、「@ｘ」、「＄１００）」、「（」、「@ｙ」、「＄２００）」、「）」、「（」、「＃ＭｉｎａｒＰｒｏｐｓ」、「（」、「@ｌａｂｅｌ」、「＄Ｃｏｍｍａｎｄ）」、「（」、「@ｃａｌｌｂａｃｋ」、「＄ｆｏｏ）」、「）」、「）」、「）」、「（」、「＃ｉｄｅｎｔｉｆｉｅｒ」、「（」、「@ｉｄ」、「＄６５２４）」、「（」、「@ｎａｍｅ」、「＄ｃｏｍｍａｎｄ）」、「（」、「@ｃｌａｓｓ」、「＄Ｃｏｍｍａｎｄ）」、「）」、「（」、「＃Ｓｔｒｕｃｔ」、「（」、「@ｐａｒｅｎｔ」、「＄６５２２」、「）」、「）」及び「）」といったトークンの列に分解されることとなる。こうして分解されたトークンの

列は配列に格納される。

こうして、字句解析部４１による分解を終えると、次の構文解析部４２では、これらトークンが格納されている配列を受け取って、上記のように、部品の識別子、部品のクラス名、部品の名前、リソースリスト、及びプログラムの中で使われる変数名、の各フィールドを持つ第４図（Ｃ）に示されるようなデータテーブルを記憶装置３上に生成する。

ここで、該構文解析部４２による上記変数名の生成手順について、第５図を併せ参照しつつ説明する。

先にも述べたように、従来のソースコード生成装置においても、こうした変数名を自動生成する機能はあったが、これらは何れも、通常人間が使用しないような文字列の組合せを選んで変数名とすることから、確かに変数名の重複はないものの、その可読性は著しく低いものとなっていた。

そこでこの実施例のソースコード生成装置では、上記部品の名前とＧＵＩデータ編集時に設けられた上記識別子とを有効利用して、構文解析部４２を通じた以下に列記する手順によって、可読性の高い変数名を自動生成するようにしている。

すなわち構文解析部４２では、

(1) 上記配列を受け取るとまず、部品の識別子を決定するために、属性の名前「ｉｄ」の値を取り出し、これを当該部品の識別子として上記データテーブルの「識別子」のフィールドに格納する。第４図に示す例においては、値「６５２４」が取り出され、これが「識別子」のフィールドに格納される。

(2) 次いで、同配列から属性の名前「ｃｌａｓｓ」の値を取り出し、これを当該部品のクラス名として上記データテーブルの「クラス名」のフィールドに格納する。第４図に示す例においては、値「Ｃｏｍｍａｎｄ」が取り出され、これが「クラス名」のフィールドに格納される。

(3) 同様に、配列から属性の名前「ｎａｍｅ」の値を取り出し、これを当該部品の名前として上記データテーブルの「部品の名前」のフィールドに格納する。第４図に示す例においては、値「ｃｏｍｍａｎｄ」が取り出され、これが「部品の名前」のフィールドに格納される。

といった一連の処理の後、同データテーブルの「変数名」のフィールドに格納すべく値を生成するために、同第４図に示す例においては、上記属性の名前「ｎａｍｅ」の値である「ｃｏｍｍａｎｄ」と、上記属性の名前「ｉｄ」の値である「６５２４」とを、該構文解析部４２内に定義されている変数名生成関数に渡す。

これにより変数名生成関数では、

(4) convertedID=Convert(id) を呼び、上記識別子「ｉｄ」の値「６５２４」を文字列に変換してこれを変数convertedIDに代入する（第５図ステップＳ１１）。

(5) 次に、strcat(widgetName,"＿")を呼び、変数widgetNameに上記部品の名前「ｎａｍｅ」の値「ｃｏｍｍａｎｄ」に応じた「ｃｏｍｍａｎｄ＿」を代入する。すなわち、こうした文字列結合関数によって、「部品の名前」の後にセパレータ「＿」が付加されるかたちとなる（第５図ステップＳ１２）。

(6) 最後に、文字列結合関数strcat(widgetName,convertedID)を呼び、これら代入した文字列「ｃｏｍｍａｎｄ＿」及び「６５２４」を結合して、変数名「ｃｏｍｍａｎｄ＿６５２４」を生成する（第５図ステップＳ１３）。

といった手順にて変数名を生成し、この生成した変数名「ｃｏｍｍａｎｄ＿６５２４」を構文解析部４２に返す。

構文解析部４２では、こうして返された変数名「ｃｏｍｍａｎｄ＿６５２４」を、上記生成するデータテーブルの「変数名」のフィールドに加えて（格納して）、その処理を終える。

このように、この実施例によれば、部品の名前と部品の識別子とをセパレータにて結合してオブジェクトプログラムで使用する変数名を生成するようにしていることから、自動生成されたソースコードを後に人為的に編集する場合でも、そこで使われている変数名の用途等については、容易に

理解することができるようになる。また、変数名の一部として、部品毎に各別の値として付される識別子を用いていることから、同一の変数名が重複して生成されることもない。

こうして構文解析部４２を通じてデータテーブルが生成されると、コード生成部４３では、記憶装置３上に用意されている上記テンプレート３１を用いて、この生成されたデータテーブルに応じたＣ言語ソースコードを生成する。

コード生成部４３は、第６図に示すように、ｃａｌｌｂａｃｋサブコード生成部４３１、ｒｅｓｏｕｒｃｅサブコード生成部４３２、ｉｎｃｌｕｄｅサブコード生成部４３３、ｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４、及びｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５の５つのサブコード生成部と、これらサブコード生成部４３１～４３５を通じて生成されるサブコードを所定に結合して、上記所望とされるソースコードを最終的に出力するコードマージ部４３０と、を具えて構成される。

ｃａｌｌｂａｃｋサブコード生成部４３１は、記憶装置３上に用意されたテンプレート３１のうちの第１テンプレート３１ａに記述されている情報に基づき、上記構文解析部２４にて生成されたデータテーブル（第４図（Ｃ）参照）から、ｃａｌｌｂａｃｋ（コールバック）関数のプログラムのみについてそのソースコードを生成する部分であり、ｒｅｓｏｕｒｃｅサブコード生成部４３２は、同テンプレート３１のうちの第２テンプレート３１ｂに記述されている情報に基づき、同データテーブルから、ｒｅｓｏｕｒｃｅ（リソース）のプログラムのみについてそのソースコードを生成する部分であり、ｉｎｃｌｕｄｅサブコード生成部４３３は、同テンプレート３１のうちの第３テンプレート３１ｃに記述されている情報に基づき、同データテーブルから、ｉｎｃｌｕｄｅ（インクルード）のプログラムのみについてそのソースコードを生成する部分であり、ｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４は、同テンプレート３１のうちの第４テンプレート３１ｄに記述されている情報に基づき、同データテーブルから、ｗｉｄｇｅｔ（ウィジェット：部品）変数の宣言のプログラムのみについてそのソースコードを生成する部分であり、そしてｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５は、同テンプレート３１のうちの第５テンプレート３１ｅに記述されている情報に基づき、同データテーブルから、部品の生成に関するプログラムのみについてそのソースコードを生成する部分である。これら各サブコード生成部によるソースコード生成手順を第７図に示す。

すなわち、上記第１～第５テンプレート３１ａ～３１ｅは、前述のように、各々割り当てられたプログラムについて、そのソースプログラムに直接出力される情報とこれに補充する必要のある情報とが所定の形式で記述されたファイルであり、これらサブコード生成部４３１～４３５では何れも、各々対象となるテンプレートへの記述情報に基づき、また、上記生成されたデータテーブルに基づき、以下に列記する手順をもって、その割り当てられたソースコード（サブコード）を生成する。

(1) まず、対象となるテンプレートを読み込む（第７図ステップＳ２１）。

(2) この読み込んだテンプレートに記述されている情報を解釈して、補充しなければならない情報を生成するのに必要なデータをリストアップする（第７図ステップＳ２２）。

(3) 上記データテーブルから該当するデータを取り出す（第７図ステップＳ２３）。

(4) この取り出したデータを、上記テンプレートに記述されている情報に基づき、そこで指定されている形式の情報に変換する（第７図ステップＳ２４）。

(5) この変換した情報を、当該テンプレートの指定されている部分に埋め込む（第７図ステップＳ２５）。

第８図は、上記各テンプレート３１ａ～３１ｅについてその具体例を例示したものであり、また

第９図は、これらテンプレート３１ａ～３１ｅに基づく上記各サブコード生成部４３１～４３５のソースコード（サブコード）生成出力例を示したものであり、以下、これら第８図及び第９図を対比しつつ、上記各サブコード生成部によるソースコード生成処理を更に具体的に説明する。なお、この例においては、上記テンプレートに記述される情報のうち、補充しなければならない情報についてはこれを、文字列「<$」と文字列「$>」とによって囲んで定義するものとする。

いま、構文解析部４２によって生成された上記データテーブルがコード生成部４３に渡されたとすると、ｃａｌｌｂａｃｋサブコード生成部４３１は、第８図（ａ）に示される内容を有する第１テンプレート３１ａを読み込んで、コールバック関数のプログラムについてそのソースコードの生成を開始する。

この際、該ｃａｌｌｂａｃｋサブコード生成部４３１では、上記データテーブル（第４図（Ｃ）参照）の「リソースリスト」フィールドから属性の名前「ｘ」とその属性値「１００」、属性の名前「ｙ」とその属性値「２００」、属性の名前「ｌａｂｅｌ」とその属性値「Ｃｏｍｍａｎｄ」、及び同部品の識別名「ｃｏｍｍａｎｄ」をそれぞれ取り出し、上記第２テンプレート３１ｂの

```
<$ Identifier _name $>
```

の部分に、識別名「ｃｏｍｍａｎｄ」を、

```
<$ resource _name $>
```

の部分に、上記各属性の名前「ｘ」、「ｙ」、「ｌａｂｅｌ」を、また

```
<$ resource _value $>
```

の部分に、上記各属性の値「１００」、「２００」、「Ｃｏｍｍａｎｄ」をそれぞれ埋め込む。そして、この第２テンプレート３１ｂにこれら各要素の名前「ｃａｌｌｂａｃｋ」の属性値「ｆｏｏ」を取り出し、上記第１テンプレート３１ａのコールバック関数埋め込み指定部分である

```
<$ func _name $>
```

の部分に、この取り出した「ｆｏｏ」を埋め込む。そして、この第１テンプレート３１ａに「ｆｏｏ」を埋め込み生成したコードを、「ｃａｌｌｂａｃｋサブコード」としてコードマージ部４３０に出力する（第９図（ａ）参照）。

また、同データテーブルがコード生成部４３に渡されると、ｒｅｓｏｕｒｃｅサブコード生成部４３２は、第８図（ｂ）に示される内容を有する第２テンプレート３１ｂを読み込んで、リソースのプログラムについてそのソースコードの生成を開始する。

この際、該ｒｅｓｏｕｒｃｅサブコード生成部４３２では、上記データテーブル（第４図（Ｃ）参照）の「リソースリスト」フィールドから属性を埋め込み生成したコードを、「ｒｅｓｏｕｒｃｅサブコード」としてコードマージ部４３０に出力する（第９図（ｂ）参照）。

同じく、コード生成部４３にデータテーブルが渡されると、ｉｎｃｌｕｄｅサブコード生成部４３３は、第８図（ｃ）に示される内容を有する第３テンプレート３１ｃを読み込んで、インクルードプログラムについてそのソースコードの生成を開始する。

この際、該ｉｎｃｌｕｄｅサブコード生成部４３３では、同データテーブル（第４図（Ｃ）参照）の「クラス名」フィールドから当該部品のクラス名「Ｃｏｍｍａｎｄ」を取り出し、上記第３テンプレート３１ｃの

```
<$ Header Widget_Class $>
```

の部分に、この取り出したクラス名「Ｃｏｍｍａｎｄ」を埋め込む。そして、この第３テンプレート３１ｃに「Ｃｏｍｍａｎｄ」を埋め込み生成し

たコードを、「ｉｎｃｌｕｄｅサブコード」としてコードマージ部４３０に出力する（第９図（ｃ）参照）。

同じく、ｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４は、データテーブルがコード生成部４３に渡されると、第８図（ｄ）に示される内容を有する第４テンプレート３１ｄを読み込んで、ｗｉｄｇｅｔ変数の宣言のプログラムについてそのソースコードの生成を開始する。

この際、該ｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４では、同データテーブル（第４図（Ｃ）参照）の「変数名」フィールドから部品を保持する変数名「ｃｏｍｍａｎｄ＿６５２４」を取り出し、上記第４テンプレート３１ｄの

```
<$ variable _name $>
```

の部分に、この取り出した変数名「ｃｏｍｍａｎｄ＿６５２４」を埋め込む。そして、この第４テンプレート３１ｄに「ｃｏｍｍａｎｄ＿６５２４」を埋め込み生成したコードを、「ｗｉｄｇｅｔ＿ｖａｒサブコード」としてコードマージ部４３０に出力する（第９図（ｄ）参照）。

そして、ｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５は、データテーブルがコード生成部４３に渡されると、第８図（ｅ）に示される内容を有する第５テンプレート３１ｅを読み込んで、部品の生成のプログラムについてそのソースコードの生成を開始する。

この際、該ｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５では、同データテーブル（第４図（Ｃ）参照）の「クラス名」のフィールドから当該部品のクラス名「Ｃｏｍｍａｎｄ」を、「部品の名前」のフィールドから当該部品の名前「ｃｏｍｍａｎｄ」を、「変数名」フィールドから当該部品の変数名「ｃｏｍｍａｎｄ＿６５２４」をそれぞれ取り出すとともに、同データテーブルの「リソースリスト」のフィールドの属性名「ｐａｒｅｎｔ」の属性値によって案内されるデータテーブル、すなわち前記（２）式にて表記されると

する親部品のオブジェクトから第４図に示した例と同様に生成されるデータテーブル（図示せず）の「変数名」のフィールドから親部品の変数名「ｓｈｅｌｌ＿６５２２」を取り出し、これら取り出した各データ要素を上記第５テンプレート３１ｅの各々指定される部分に埋め込む。なお、第８図（ｅ）に示すこの第５テンプレート３１ｅにおいて、文字列「$$」で囲まれた部分は条件式であって、この場合、「ｃｌａｓｓ」が「ｓｈｅｌｌ」である場合には、前半の

```
    XtAppContext app_con;
    <$ variable _name $> = XtAppInitialize(
&app_con, "<$ widget _name $>",NULL, ZERO,
 &argc, argv, fallback_resources, NULL, ZER
O);
```

と記述された部分を使い、それ以外の場合には、後半の

```
    <$ variable _name $> = XtCreateManagedW
idget("<$ widget_name $>", <$ widget _clas
s $>WidgetClass, <$ parent_variable $>, NUL
L,ZERO);
```

と記述された部分を使うことを意味する。ここでの例では、「ｃｌａｓｓ」が「Ｃｏｍｍａｎｄ」となっていることから、後者の部分が使われる。したがってこの例の場合、ｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５では、該第５テンプレート３１ｅにおける後半部分の

```
<$ variable _name $>
```

の部分に、当該部品の変数名「ｃｏｍｍａｎｄ＿６５２４」を、

```
<$ widget _name $>
```

の部分に、当該部品の名前「ｃｏｍｍａｎｄ」を、

また

```
<$ widget _class $>
```

の部分に、当該部品のクラス名「Ｃｏｍｍａｎｄ」を、そして

```
<$ parent _variable $>
```

の部分に、親部品の変数名「ｓｈｅｌｌ＿６５２２」をそれぞれ埋め込む。そして、この第５テンプレート３１ｅにこれら各要素を埋め込み生成したコードを、「ｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード」としてコードマージ部４３０に出力する（第９図（ｅ）参照）。

これら各サブコード生成部による以上のサブコード生成処理は、前記（２）式にて表記されるとした親部品のオブジェクトに対しても同様に実施される。

これらの各サブコードが入力されるコードマージ部４３０は、上述のように、これらサブコードを所定に結合して、上記所望とされるソースコードを出力する部分であり、具体的には、以下の手順によって、第１０図に示されるようなＣ言語のソースプログラムを最終的に生成し、この生成したコードを記憶装置３に出力する。

(1) ｉｎｃｌｕｄｅサブコード生成部４３３によって生成された第９図（ｃ）に示されるｉｎｃｌｕｄｅサブコードと、ｃａｌｌｂａｃｋサブコード生成部４３１によって生成された第９図（ａ）に示されるｃａｌｌｂａｃｋサブコードとをマージして、第１０図（ａ）に示されるようなコールバック関数のソースプログラムを生成する。

(2) この生成したソースプログラムを記憶装置３に出力する。

(3) ｉｎｃｌｕｄｅサブコード生成部４３３によって生成された第９図（ｃ）に示されるｉｎｃｌｕｄｅサブコードと、ｒｅｓｏｕｒｃｅサブコード生成部４３２によって生成された第９図（ｂ）に示されるｒｅｓｏｕｒｃｅサブコードと、ｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４によって生成された第９図（ｄ）に示されるｗｉｄｇｅｔ＿ｖａｒサブコードと、ｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３５によって生成された第９図（ｅ）に示されるｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコードとをマージするとともに、Ｃ言語の予約語である

```
main(argc, argv)
int argc;
char **argv;
{

}
```

を追加して、第１０図（ｂ）に示されるような部品（Ｗｉｄｇｅｔ）を生成するプログラムのソースコードを生成する。

(4) この生成したソースプログラムを記憶装置３に出力する。

なお、第１０図（ｂ）に示すソースプログラムにおいて、

```
Widget shell_6522;
```

は、上記親部品のオブジェクトに関してｗｉｄｇｅｔ＿ｖａｒサブコード生成部４３４が生成したｗｉｄｇｅｔ＿ｖａｒサブコードの部分であり、また

```
XtAppContext app_con;

shell_6522 = XtAppInitialize(&app_con,
"shell", NULL, ZERO, &argc, argv, fallback_resources, NULL, ZERO);
```

は、同じく上記親部品のオブジェクトに関してｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコード生成部４３４が生成したｗｉｄｇｅｔ＿ｃｒｅａｔｅサブコ

ードの部分である。

このように、この実施例によるソースコード生成装置では、上記コード生成部４３を、生成対象とするソースコードの各々異なる機能部分について各別にコード生成を行う複数の（上記の例では５つの）サブコード生成部と、これら各サブコード生成部にて生成される各別のコードを、対象とするプログラミング言語の仕様に応じて結合するコードマージ部とを具えて構成するとともに、上記テンプレート３１も、これらサブコード生成部の別に、各々それに見合ったものを複数用意するようにしたことから、上記のように、ソースコードの行の並びに対して独立に所望のコードを生成して、これを任意に結合することができるようになる。このため、ＧＵＩ環境や生成の対象となるソースプログラムの言語仕様が異なる場合であっても、具体的には、ソースプログラムに埋め込まれるテキストの順の等が異なる場合であっても、従来のように、これらＧＵＩ環境や言語仕様の別にソースコード生成装置を用意する必要はなくなり、該ソースコード生成装置としての汎用性も著しく高められるようになる。

なお、上記の実施例では便宜上、Ｘウィンドウシステム・バージョン１１の、Ａｔｈｅｎａ　Ｗｉｄｇｅｔ（アテナ・ウィジェット）を使用し、また生成の対象とするソースコードとしては、Ｃ言語で記述されたソースプログラムとする場合について説明したが、この発明にかかるソースコード生成装置が、こうした環境やＯＳ（オペレーティングシステム）、更には言語仕様に限定されるものでないことは勿論である。

また、このソースコード生成装置において読み込み対象とするＧＵＩ編集データも、必ずしも前述したＳ式を用いて表記する必要はない。要は、当該ＧＵＩデータをオブジェクトの集合として捉えてこれらを所要に分類でき、その各要素の属性等を的確に表すことのできる表記方法であれば、すなわち、その字句解析並びに構文解析を通じて、第４図（Ｃ）に例示した態様に準じた形でのデータテーブル生成を可能とする表記方法であればよい。ただしこの場合、その表記方法と字句並びに構文解析方法との対応を新たに定義する必要はある。

［発明の効果］

以上説明したように、この発明によれば、生成対象とするソースコードとして直接に付加される文字列、及び同生成対象とするソースコードとするために構文解析された内容に対して補う必要のある情報が記述されたテンプレートを設け、このテンプレートを通じて、異なるＧＵＩ構築環境間における異なるコード生成要素を吸収するようにしたことから、各種のＧＵＩ構築環境に対し汎用的に、目的とされるプログラムのソースコードを生成することができるようになる。

４．図面の簡単な説明

第１図は、この発明にかかるソースコード生成装置の一実施例をＧＵＩ編集装置とともに示すブロック図である。

第２図は、ＧＵＩ編集装置における編集画面の一例を示す平面図である。

第３図は、上記のＧＵＩ編集装置で編集され、上記実施例ソースコード生成装置に読み込まれるＧＵＩ部品情報の管理構造について、その一例を模式的に示す略図である。

第４図は、同実施例ソースコード生成装置による字句解析処理及び構文解析処理を通じて生成されるトークンの列及びデータテーブルについて、その一例を模式的に示す略図である。

第５図は、同実施例ソースコード生成装置による変数名生成手順を例示するフローチャートである。

第６図は、同実施例ソースコード生成装置のコード生成部について、その具体構成と記憶装置との係わりを示すブロック図である。

第７図は、上記コード生成部を構成する各サブコード生成部のソースコード（サブコード）生成手順を示すフローチャートである。

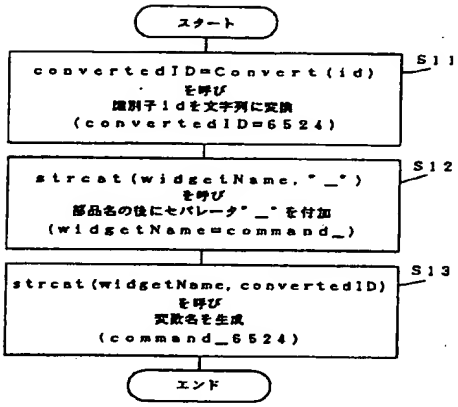
第８図は、上記記憶装置上に用意されるテンプレートの具体例を示す略図である。

第９図は、この第８図に示されるテンプレートに基づき上記各サブコード生成部が生成するサブコード例を示す略図である。

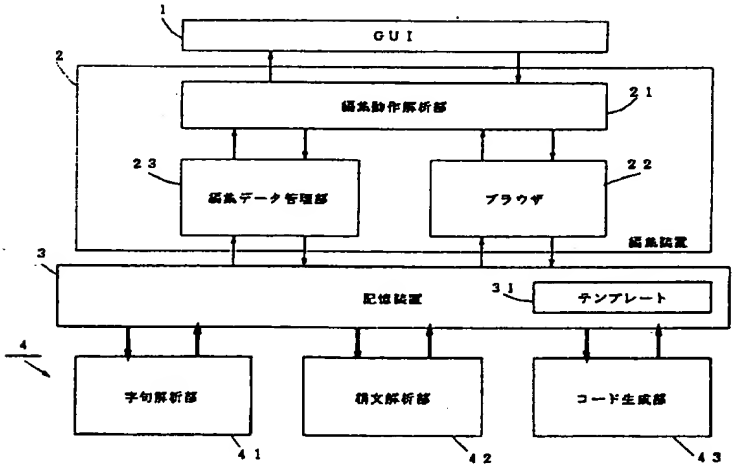
第１０図は、この実施例ソースコード生成装置によって最終的に生成出力されるソースコード（ソースプログラム）の一例を示す略図である。

１...ＧＵＩ、２...編集装置、２１...編集動作解析部、２２...ブラウザ、２３...編集データ管理部、３...記憶装置、３１...テンプレート、４...ソースコード生成装置、４１...字句解析部、４２...構文解析部、４３...コード生成部、４３０...コードマージ部、４３１～４３５...サブコード生成部。

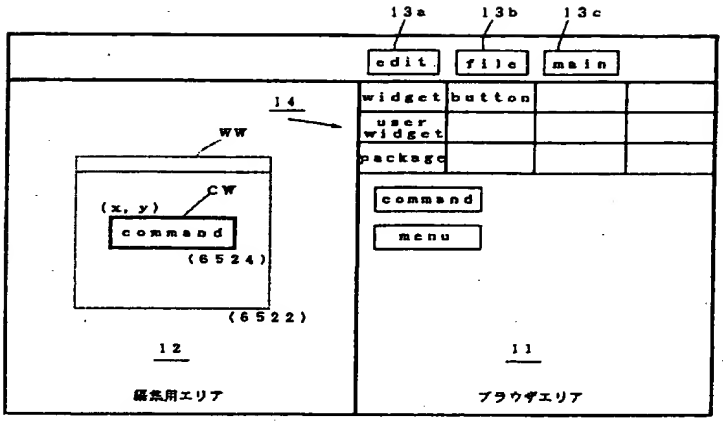
出願人代理人　木村高久



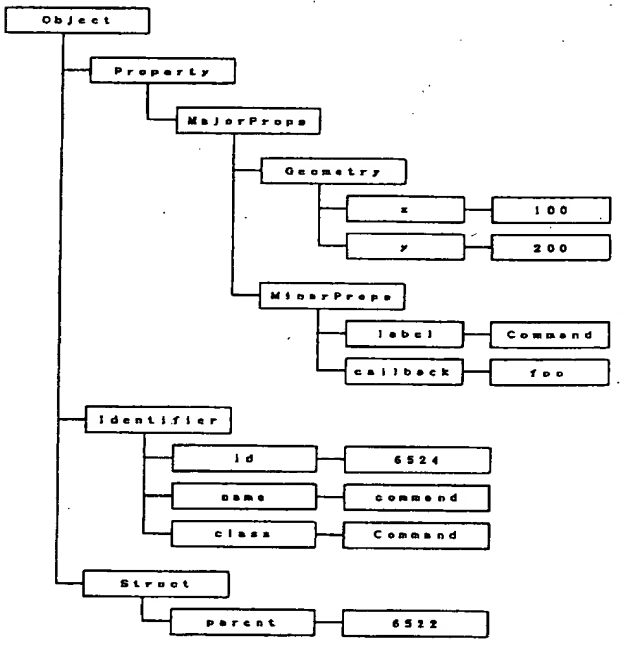
第５図



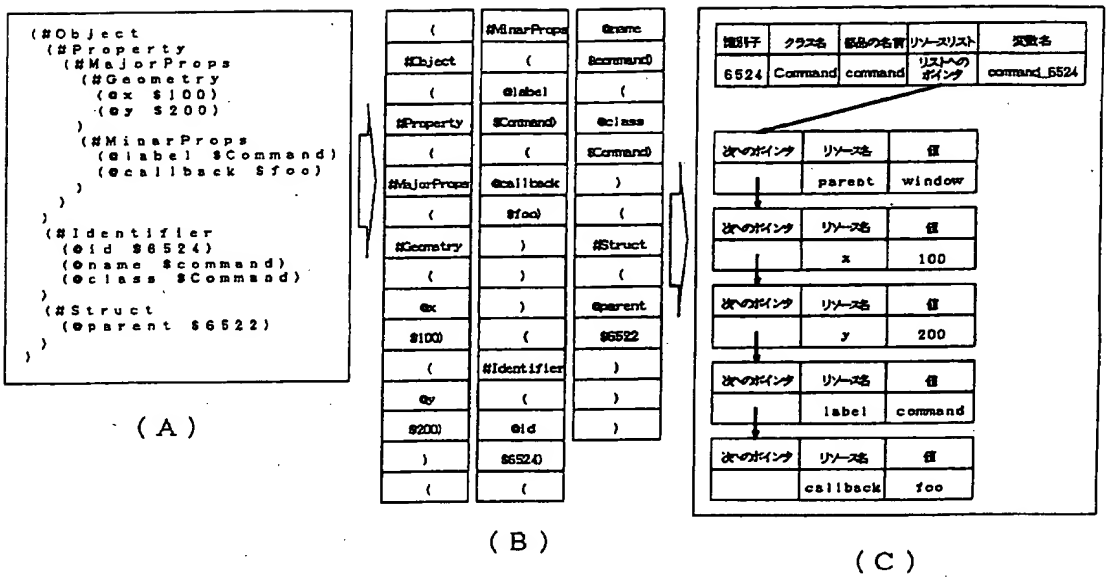
第１図


13a
13b
13c
edit
file
main
widget
button
user widget
package
14
WW
(x, y)
CW
command
(6524)
(6522)
command
menu
12
11
編集用エリア
ブラウザエリア

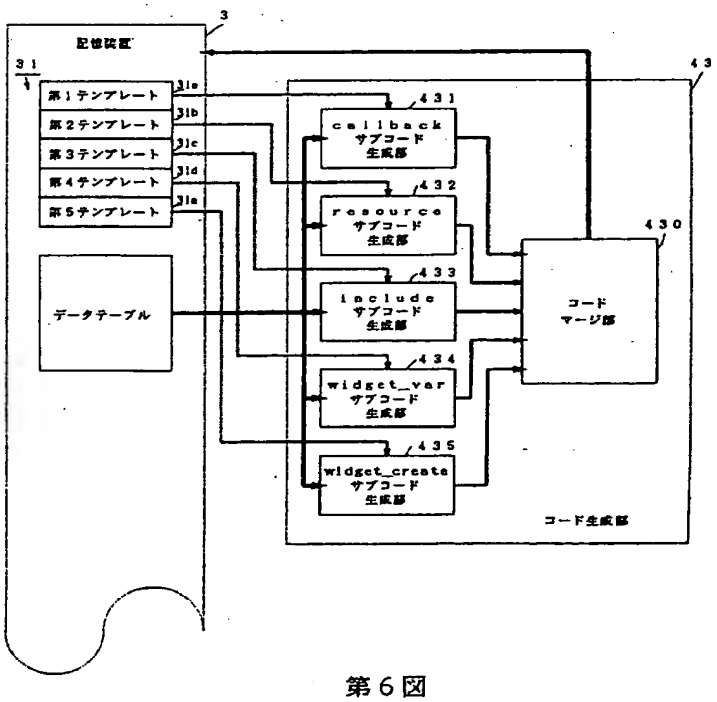
第２図


Object
Property
MajorProps
Geometry
x
100
y
200
MinorProps
label
Command
callback
foo
Identifier
id
6524
name
command
class
Command
Struct
parent
6522

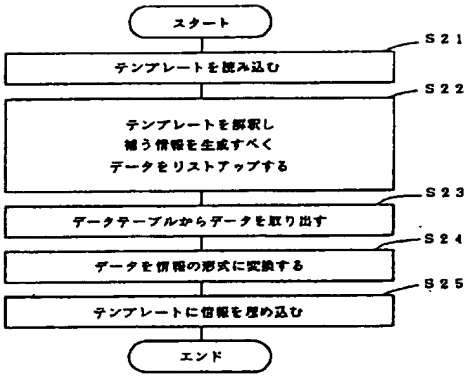
第３図


(#Object
(#Property
(#MajorProps
(#Geometry
(@x $100)
(@y $200)
)
(#MinarProps
(@label $Command)
(@callback $foo)
)
)
)
(#Identifier
(@id $6524)
(@name $command)
(@class $Command)
)
(#Struct
(@parent $6522)
)
)
識別子
クラス名
部品の名前
リソースリスト
変数名
6524
Command
command
リストへのポインタ
command_6524
次へのポインタ
リソース名
値
parent
window
x
100
y
200
label
command
callback
foo
（A）
（B）
（C）

第４図


3
記憶装置
31
第１テンプレート
第２テンプレート
第３テンプレート
第４テンプレート
第５テンプレート
31a
31b
31c
31d
31e
データテーブル
43
431
callback
サブコード
生成部
432
resource
サブコード
生成部
433
include
サブコード
生成部
434
widget_var
サブコード
生成部
435
widget_create
サブコード
生成部
430
コード
マージ部
コード生成部

第６図



第７図

callback関数のプログラム

```
#include   < X11/Command.h >

    void foo(w, client_data,call_data)
        Widget w;
        caddr_t client_data;
        caddr_t *call_data;
    (
        fprintf(stdout, "called");
    )
```

第１０図（ａ）

（ａ） 第1テンプレート31a

```
void <$ func_name $>(w, client_data, event)
    Widget  w;
    caddr_t client_data;
    caddr_t *call_data;
(
    fprintf(stdout, "called");
)
```

（ｂ） 第2テンプレート31b

```
String fallback_resources[] = {
"*<$ Identifier_name $>.XtN<$ resource_name $>, <$ resource_value $>",
NULL
};
```

（ｃ） 第3テンプレート31c

```
#include    "X11/<$ Header Widget_Class $>.h"
```

（ｄ） 第4テンプレート31d

```
Widget <$ variable_name $>;
```

（ｅ） 第5テンプレート31e

```
$$IF (class == shell) $$
    XtAppContext app_con;
    <$ variable_name $> = XtAppInitialize(&app_con, "<$ widget_name
    $>",NULL, ZERO, &argc, argv, fallback_resources, NULL, ZERO);

$$ELSE$$
    <$ variable_name $> = XtCreateManagedWidget("<$ widget_name $>",
    <$ widget_class $>WidgetClass, <$ parent_variable $>, NULL,ZERO);

$$ENDIF$$
```

第８図

(a) callbackサブコード生成出力

```
void foo(w, client_data, event)
    Widget w;
    caddr_t client_data;
    caddr_t *call_data;
{
    fprintf(stdout, "called");
}
```

(b) resourceサブコード生成出力

```
String fallback_resources[] = {
"*command.XtNx, 100",
"*command.XtNy, 200",
"*command.Xlabel, Command",
NULL
};
```

(c) includeサブコード生成出力

```
#include    < X11/Command.h >
```

(d) widget_varサブコード生成出力

```
Widget command_6524;
```

(e) widget_createサブコード生成出力

```
command_6524 = XtCreateManagedWidget("command", CommandWidgetClass,
shell_6522, NULL, ZERO);
```

第９図

部品（Widget）を生成するプログラム

```
#include    < X11/Command.h >

String fallback_resources[] = {
"*command.XtNx, 100",
"*command.XtNy, 200",
"*command.Xlabel, Command",
NULL
};

main(argc, argv)
int argc;
char **argv;
{

Widget shell_6522;

Widget command_6524;

XtAppContext app_con;

shell_6522 = XtAppInitialize(&app_con, "shell", NULL, ZERO,
    &argc, argv, fallback_resources, NULL, ZERO);
command_6524 = XtCreateManagedWidget("command", CommandWidgetClass,
    shell_6522, NULL, ZERO);

}
```

第１０図（ｂ）